A location aware mobile tool for direct and indirect climate data sensors

Francisco Ramos[1], Aida Monfort, Joaquin Huerta

Institute of New Imaging Technologies, University Jaume I of Castellon, Spain

Abstract Access to GIS data from mobile platforms continues being a challenge and there exists a wide range of fields where it is extremely useful. In this work, we combined three key aspects: climate data sensors, mobile platforms and spatial proximity operations. We published and made use of a web 2.0 network of climate data, where content is user-collected, by means of their meteorological stations, and exposed as available information for the virtual community. Moreover, we enriched this data by giving the users the opportunity to directly inform the system with different climate measures. In general, management of this type of information from a mobile application could result in an important decision tool, as it enables us to provide climate-related data according to a context and a geographical location. Therefore, we implemented a native mobile application for iPhone and iPad platforms by using ArcGIS SDK for iOS and by integrating a series of ArcGIS webmaps, which allows us to perform geospatial queries based on the user's location, offering, at the same time, access to all the data provided by the climate data sensor network and from direct users.

1 Introduction

Nowadays, access to data sensors is progressively being adapted to different standards with a clear objective: to increase their interoperability and visibility. In this context, the growth of volunteered geographic information [1] as sensor data providers is an assertive method of collecting geospatial information, and it seems difficult to force the use of strict standards in this kind of data workflow. This free source of data, through the efforts of volunteers, is growing and becoming a very important base to build applications that exploit such data. Our work takes advantage of this new manner of collecting and distributing information by using users and a climate data sensor: Meteoclimatic[2], a network of meteorological stations updated in real-time and which are owned by non-professional volunteers, that is to say, a type of VGI context.

[1] Email addreses: francisco.ramos@uji.es, aida.monfort@uji.es, huerta@uji.es
[2] http://www.meteoclimatic.com/

In this work, we aimed at integrating a climate data sensor and VGI into a mobile platform for the easy management and accessibility to this type of information. Our tool features the following contributions:

- VGI mashup: we combine data from different sources in a single application. These data sources come from a network of climate data sensors: meteoclimatic, where users own meteorological stations and offer data to the virtual community, and from our application users, who could directly add information to the system at any time.
- Availability: big data [2] consist of technologies and initiatives that involve data that are too diverse, change quickly or massively for conventional technologies, skills and infrastructure to approach efficiently. In other words, the volume, speed or variety of data is enormous. In this context, we process a big data source such as a meteorological sensor, providing a service layer, which offers data in a reliable and efficient manner.
- Climate data sensors comparison: we performed a comparative study of popular and well-know climate data sensors.

As previously commented, we also consider important data visibility. Thus, the fact of creating actionable information and usable applications from data sensors offers valuable insights by exposing and exploiting the geographic dimension of that data using maps as a visualization tool to non-gis users. This application is open and visible to all those users through a platform of distribution for mobile platforms. Moreover, we also provide a scheme for such a type of mobile applications, with existing components, and a standard API to access data in a networked environment.

We have organized this paper as follows: we will carry out an analysis of related work in section 2. Section 3 presents the architecture of our tool with internal details. After that, in section 4, we introduce the implementation details with an overview of the different technologies we made use of. Later, section 5 shows the different results obtained. Finally, conclusions are exposed in section 6.

2 Related work

VGI is a term coined by Goodchild [1], used to define the web usage with the aim to create, assemble and disseminate geographic data provided voluntarily by ordinary citizens. However there are more names for the same phenomenon, as the wikification of GIS [3]. Although VGI is the most broadly used term, there is a discussion whether the word volunteered is appropriate or not [4]: not all VGI data are provided only by users, since some organizations can help in the task of getting the information. Moreover, volunteered is not always the right adjective as in some crowdsourcing geographic information systems [5], the user has to be aware

of all the information he provides and decide when to provide it for it to be volunteered, and not everybody might realize that their smartphone is reporting locational data to the Internet.

In the recent years, the market of smartphones has reached particularly high levels; this fact makes the process of sharing crowdsourcing information (either voluntarily or not) easier and faster. The level of data accuracy is increasing according to the improvements in software and hardware devices, but still there are some issues with the mobile GIS users regarding the Internet connection because it is not always possible to work connected due to noisy, limited or insecure connections [6].

VGI has had a great impact on Geographic Information Science (GIS), because data collection was usually one of the tasks that required more time and effort. If some time ago, all geographic data needed to be collected manually or localized by GIS experts, nowadays everybody can act as a creator of geographic data, since each citizen generates a big amount of data when using their smartphones.

Besides, the number of sensors embedded in these devices has also dramatically increased. The combination of these two trends has enabled a new kind of research called people centric sensing [7][8][9]. Therefore, sensors in mobile phones and other wireless devices can be used to collect large quantities of continuous measurements about their users. The different data modalities collected through the client can be categorized as follows: Social interaction data, that can be inferred from call logs, short message logs, Bluetooth scanning results, acoustic environment samples, etc.; Location data: that can be determined based on GPS (when available), cellular network information, and WLAN access point information (when available); Media creation and usage data: information can be captured concerning locations where images have been captured, video shot or music played; and Behavioural data: information can be captured concerning application usage, activity detection based on acceleration sensor, and regular device usage statistics based on call and short message logs.

In this work, we also consider two types of data sensors, direct and indirect ones. Indirect ones refer to the typical networked data sensor; direct ones refer to those data provided by users into applications. The fact of merging them can lead us to an interesting mashup of information. In Table 1, we show a direct climate data sensors comparison from different providers, at a given time. As another aim of our work, we consider interesting to compare these direct sensors to the information provided by users.

Therefore, the combination of all these elements lead us to the development of tools to share and display geographically located information that can help to solve problems or in the decision making process.

Thus, we propose a mobile application based on VGI data provided by direct and indirect sensors related to climate data. In order to achieve this aim we designed an architecture based on levels, and we published the final result in a mobile store platform.

Sensor	Location Coordinates Elevation	Date	Temp. °C	Humidity %	Wind km/h	Pressure hPa	Rain mm
	CASTELLÓN (SPAIN)						
Meteoclimatic	40° 02' 43'' N 00° 03' 45'' E 14 m	13-02-2014 17:46 UTC	23.6	41	11	1013.0	0
AEMET	39° 57' 26'' N 0° 4' 19'' O 43 m	13-02-2014 18:00 UTC	24.0	33	17	1006.6	0
OpenWeatherMaps	Not available	13-02-2014 18:50 UTC	23.2	37	14	1012.6	0
	VALENCIA (SPAIN)						
Meteoclimatic	39° 27' 42'' N 0° 23' 57'' W 45 m	13-02-2014 17:43 UTC	21.8	51	13	1015	0
AEMET	39° 28' 50'' N 0° 21' 59'' O 11m	13-02-2014 18:00 UTC	22.9	39	15	N/A	0
OpenWeatherMaps	Not available	13-02-2014 18:41 UTC	22.3	51	11	1011	0

Table 1. Climate data comparison from different providers or sensors at a given time: Meteoclimatic, AEMET and OpenWeatherMaps.

3 Architecture

As shown in Figure 1, the architecture we propose is based on three layers: VGI, data and service layers.

First layer (VGI) is composed of citizens, who might participate as direct sensors, i.e. mobile devices, or indirect sensors (see table 1), i.e. climate stations, we support and implement both. There are a lot of platforms that are used by citizens to share geographic information, and they can share information such as crime reports, buildings and streets, climate information and so on, it depends on the platform.

Second layer (data layer) is the component that provides some functionality to retrieve and store the data, usually through an API or a service. Thus, data from the corresponding sensor is offered to the next level, in this case, the data layer. In this layer, all the information provided by the data sensor is maintained in such a way that the service layer could exactly adjust the information required for the client application to consume.

Finally, the service layer is responsible for listening to the clients' requests and sending the required data. Finally, clients show the data. As shown in Figure 1,

layers interact one with each other to obtain data from or to pass data to another layer.

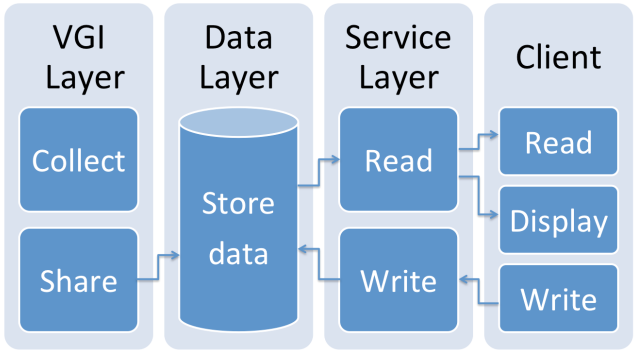


Fig. 1. Main operations between layers in a VGI platform

3.1 VGI Tools

In general, we could consider three main use cases in a VGI tool[10] as shown in Figure 2: view, report and receive information from or to the server. To view the information, the tool may display a table, a map or a collection of elements. Generally users can query or filter the information they want to observe. In order to report new data, the tool usually provides some application program interface (API) that interacts with the client to collect the data. In some cases, users can also receive alerts: when new information is provided, when new data is needed or when a specific warning occurs.

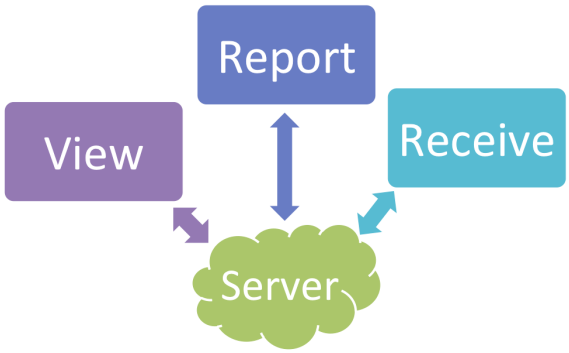
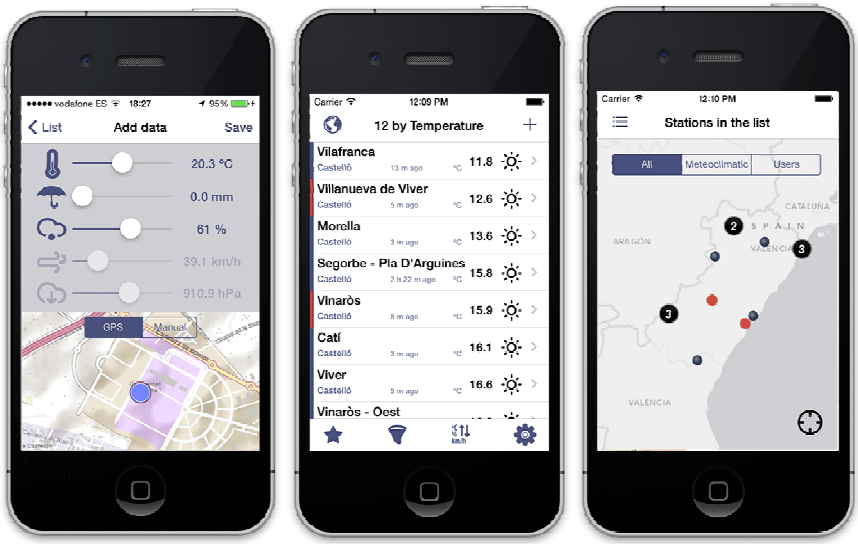


Fig. 2. View, Report and Receive are the main use cases in a mobile VGI platform.

4 Implementation

4.1 Climate data sensor

As far as the source of data of this work is concerned, and as we previously commented, we are using data directly provided by our users (see Figure 3), and a climate data sensor from a community called Meteoclimatic. This community of users provides a free service for sharing user-generated climate data. In fact, they buy their own climate stations. Therefore, these owners, or users, can register an automatic meteorological station along with the climate sensors they have, the city and the exact geographical coordinates, and they install a software that communicates with the community service, sending files with the sensors data and providing, in this way, what we can consider VGI: volunteered geographic information[1].



a) b) c)

Fig. 3. From left to right, a) users are able to enrich the system at any time with climate information, b) the application shows, in different colours, information provided by direct and indirect users, red and blue, respectively. c) map shows clustering without differentiate sensors data source but particular stations.

Nowadays, there are 1983 stations registered along Spain, Portugal and south of France. Meteoclimatic has also a web portal with real climate information from the registered stations and it also offers weather forecast, among other services.
From the implementation perspective, climate stations provide the information from their sensors by means of a RSS feed.
They provide RSS feeds to read this information. For example, in the following URL we get the information for all the stations located in Spain: *http://www.meteoclimatic.com /feed/rss/ESP*
Each station has an identifier, which can be used to get information about a specific station. For example, to read the Burriana station's feed we might use the RSS URL followed by the station identifier[3]. This RSS feed loads a web page as shown in Figure 4.

Fig. 4. RSS feed from Burriana's station

In Figure 5, we can also observe the part of the RSS related to the climate data from the sensor.

```
<!--
[[<BEGIN:ESPVA1200000012530A:DATA>]]
[[<ESPVA1200000012530A;(7,3;7,3;5,2;sun);(49,0;64,0;49,0);
(1025,1;1026,2;1024,1);(5,0;34,0;351);(0,0);Burriana>]]
[[<END:ESPVA1200000012530A:DATA>]]
-->
```

Fig. 5. Data sensor in the HTML code from the RSS feed.

In the Meteoclimatic documentation[4] we could read and understand this particular format, for example, in the second line, after the station identifier, we first obtain the values for the temperature sensor, current, maximum and minimum values respectively, then for humidity, pressure, wind and precipitation.
This part of the RSS feed is not very easy to be processed, as it is provided in a non-standard format. Thus, we needed to implement a service to decode the values

[3] http://www.meteoclimatic.com/feed/rss/ESPVA1200000012530A

[4] http://www.meteoclimatic.com/index/wp/rss_es.html

and save them into a database which we will use later to retrieve the sensor values in a better organized way through an API REST.
For each station, this service periodically updates the database with the new values for each measure.

4.2 Mobile platform

For the client application implementation, we decided to create an iOS native application due to two important aspects:

- The interaction between the user and the application is easier when the application is native; the process of distributing, installing and keeping the application in the mobile device is easier when the application comes from the official App Store. Moreover, the different controls, maps and views have less time to respond if they are native and it is easier to access the mobile device sensors, such as the GPS.
- The ArcGIS mapping platform SDK for maps we wanted to integrate had more documentation and available functions for the iOS version than for the Android version.

However, it is important to underline that our architecture could be implemented in other mobile platforms as well.
In the client, we use the device GPS when available and when the user
allows the application to gather the GPS data. By using the provided coordinates we can estimate and show the user how many kilometres are between his current position and the meteorological stations, approximately. In the mobile application, we also show, within an ArcGIS map, the stations location. As expected, the ArcGIS map is integrated with the ArcGIS SDK for iOS. With this SDK the map loads a basemap first, which can be a basemap from the available default basemaps or a user-defined basemap. Other layers are also available to be loaded or created directly in the application or it is possible to load a full webmap from the ArcGIS online platform as well.
In the ArcGIS online webpage you can basically create a webmap, add basemaps, add layers from a file or from the web and finally you can save and share the map. This shared map has an identifier, which can be used to import it to the SDK in iOS, as we performed.

4.3 API implementation

The communication between the client and the server is done through an API REST as long as the client needs to obtain or refresh the sensor values.

REST stands for Representational State Transfer and describes a set of constraints:

- Uniform interface: Interfaces are resource based; client and server interchange information usually in HTML, XML or JSON format.
- Stateless: Statelessness enables greater scalability since the server does not have to maintain, update or communicate session states, it is the client who sends the information needed in the requests body.
- Cacheable: Clients can cache Responses.
- Client-Server: Clients and servers are developed separately; this increases system portability and scalability.
- Layered System: Systems based on layers may enforce security, a client cannot know if it is connected to the server.
- Code on Demand (optional): Servers can expand the functionality of the system when required.

The client application sends some parameters to the API REST through the URL; those are used to construct a SQL query that returns the appropriate stations required by the client in JSON format. Among these parameters there are, (if available): the current device latitude and longitude so the query calculates and returns the distances. In addition, the application allows the user to specify from which province and how many stations he wants to consult, and in which order.
In Figure 6, we can examine the data in JSON format returned from the REST web service. Province filter was set to Castelló, limit to 2, and the sorting measure is temperature by ascending order.

```
{
    "data": [
        {
            "B": "1032.7",
            "distance": "4434.80739818563",
            "Elevation": "529.600463867188",
            "H": "93",
            "ID": "1424",
            "LATITUD": "39.9058",
            "LONGITUD": "-0.614167",
            "NAME": "Viver ",
            "P": "0",
            "PROVINCIA": "Castelló",
            "Rain": "gebre",
            "T": "-4.2",
            "Tmax": "-2.7",
            "Tmin": "-4.3",
            "UPDATED": "2013-12-10 06:17:00",
            "W": "0"
        },
        {
            "B": "1032.4",
            "distance": "4497.14258236582",
            "Elevation": "664.73583984375",
            "H": "86",
            "ID": "188",
            "LATITUD": "40.4719",
            "LONGITUD": "0.0216667",
            "NAME": "Catí ",
            "P": "0.4",
            "PROVINCIA": "Castelló",
            "Rain": "hazesun",
            "T": "8.7",
            "Tmax": "12.9",
            "Tmin": "-0.4",
            "UPDATED": "2013-12-10 17:19:00",
            "W": "3"
        }
    ]
}
```


Fig. 6. Web service result in JSON format example.

In Figure 7, we show the final architecture and technologies used in this work.

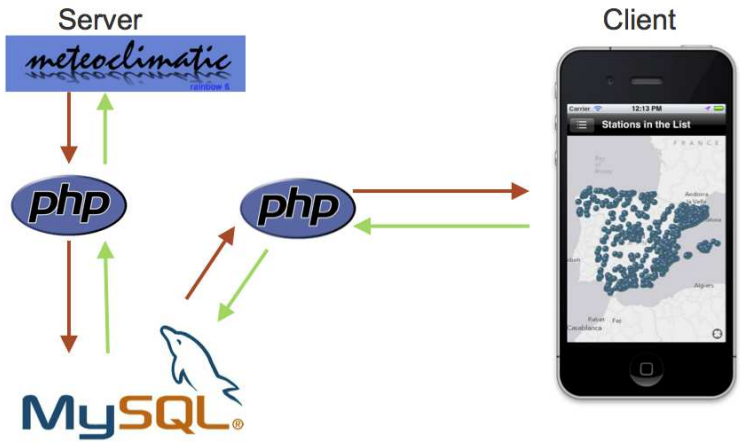


Fig. 7. Final architecture and technologies used in this work.

As for the service that accesses Meteoclimatic sensor data and updates the database, it was implemented in php. For the database, we used MySQL, for the API REST that gets the parameters from the client, queries the database and returns the results, we used a RESTful Web application PHP library such as Tonic[5]. Finally, the client was implemented in Objective-C using the XCode integrated development environment.

4.4 Maps backend

As mentioned above, about GIS technology, we are using ArcGIS online platform to create the map, and ArcGIS SDK for iOS to integrate the map in the native application. There are some options to show the points of the stations on the map view in the application, and these are the ones we decided to use:

- KML[6] Layer.
- CSV[7] Layer
- Graphics Layer created in client application.

Our first attempt was to create a webmap in ArcGIS online, with a KML layer containing all the stations. As shown in Figure 8, the KML generated was correctly created. However, due to existing limitations in the ArcGIS SDK for iOS, the layer failed to appear.

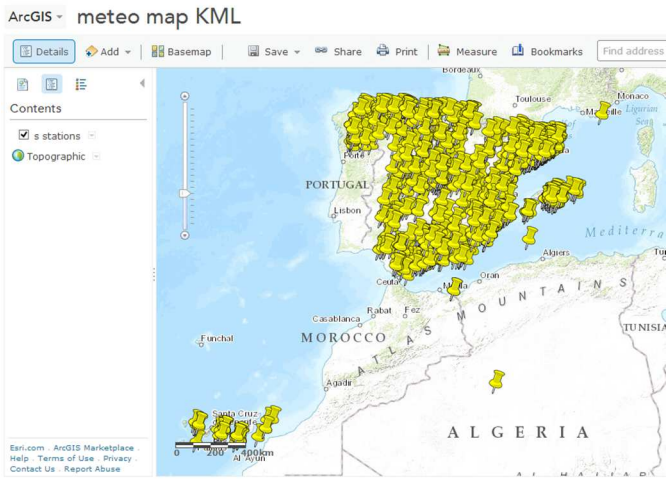


Fig. 8. ArcGIS online webmap with KML Layer imported from the server.

5 http://www.peej.co.uk/tonic/

6 Keyhole Markup Language

7 Comma-separated Values

Our second choice was using a CSV imported file. As shown in Figure 9, it worked in the iOS integrated map, but as CSV files are static files, it was difficult to refresh the layer when we updated the file in our server. It is important to underline that our RESTful services are dynamic, that is, they dynamically obtain the information from the data sensor provider as explained in section 4.3.

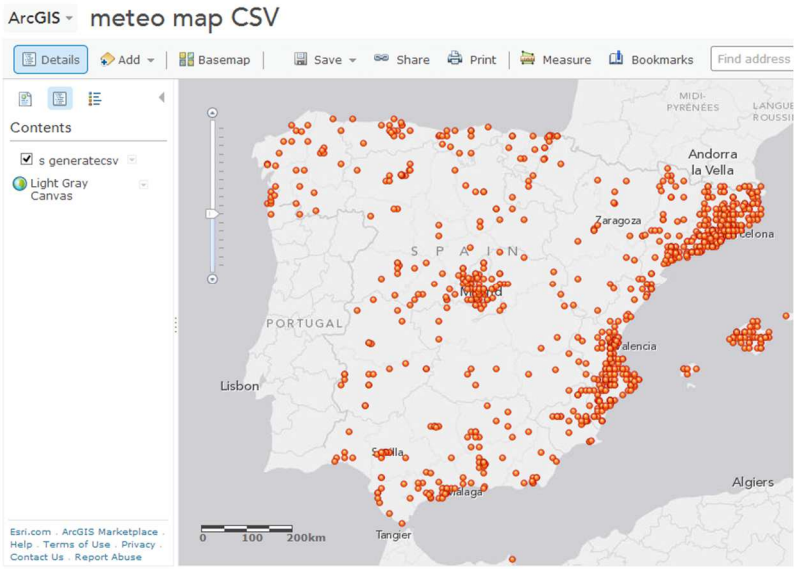


Fig. 9. ArcGIS online webmap with CSV Layer imported from server.

Therefore, we decided to add a graphics layer directly in the application and paint the desired points in that layer whenever it was needed.

At this point, some cluster analysis was needed to make the map interface clearer to understand.

Clustering[11] is the task of grouping some elements by some given properties. These groups are called clusters. Elements in the same cluster are more similar among them, than elements from other cluster. For example, when working with population data, individuals could be clustered by age, location or other attributes.

In our case, we needed to group stations by its location on the map, most of mapping services do cluster analysis automatically, because the data are displayed clearly, and it helps in the map reading and interpreting process.

Unfortunately, there is not an available clustering process in iOS SDK, so we implemented a module in the client application to calculate and redraw the points grouped in clusters by a proximity criterion. This calculation depends on some parameters:

- Points latitude and longitude coordinates.
- Map zoom level.
- Current extent on the map.

With these parameters the clustering module obtains the points coordinates that appear on the screen, which are inside the current extent, and calculates the position for the clusters, and how many points are grouped inside. Every time the user zooms in, zooms out or changes the extent of the map, this process is repeated to get the new clusters positions. In Figure 10, we show both versions of the map view, the one on the left, before the clustering module was developed, and the one on the right, with the resulting clusters calculated by our clustering module.

Fig. 10. From left to right, map view without and with clustering analysis, respectively.

5 Results

5.1 Deployment

In Figure 11, we can observe the details of the three versions of the mobile application (named Meteo Reader) submitted to App Store. First version of the iOS App was submitted on 14th August and was released on 20th August.
We submitted a new version[8] on 16th September with software integrated to track users, user sessions and time spent by users on the application.
With the iOS7 release, for iOS devices, we submitted a new version according to the new iOS aesthetic and with some improvements, such as showing the station

[8] Meteo v1.1 on simulator http://youtu.be/sXhZ7T6yeHM

elevation together with the sensors data. This version was submitted on 4th October and was released on 11th October.

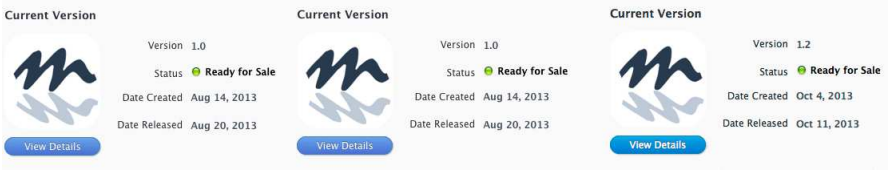


Fig. 11. METEO READER versions in the App Store.

5.2 Tracking users and sessions

Tracking software, which was added in the second version, consists of a Software Development Kit (SDK), which is integrated in the application. It provides us with interesting information about the application usage. We are not storing users data, that is, users can use our application without being registered. Therefore, we only had the sales and download information provided by default within iTunes developers platform, which does not include any information about the actual usage data. For this reason, we integrated this software into the application with the aim of monitoring, tracking users, and obtaining some useful information such as usage patterns, user retention and errors occurred in the application. The SDK we are using is called Flurry and it is available to download in their webpage[9]. This SDK is available for most used mobile platforms like iOS, Android or Windows Phone. With the Flurry SDK integrated in the application from version 1.1 (second version) we obtained very interesting data about the users behaviour. Every time users send the application to the background by pressing the home button, Flurry sends data to their system, that we can later analyse on their webpage. These data include user location, time spent on the application, application version and even error or crash reports when they occur.

The actual usage data summary of Flurry information contains how many sessions there are, the median session length, how many new users there are and the average of active users, as shown in Figure 12.

[9] http://www.flurry.com/

Fig. 12. General information about sessions and users from 16th September to 3rd December 2013.

Chart shown in Figure 13 is related to the number of sessions per week. There are 3982 sessions in total, distributed from 16th September until the 3rd December. First weeks, increasing trend of sessions is very high due to users updating the application to the new version with Flurry, but still there is an interesting usage of the application with an average of more than 250 sessions per week.

Fig. 13. Number of sessions per week, from 16th September to late November (sessions for the last week are still being calculated).

We also obtained a valuable feedback from the chart of session frequency (Figure 14). This chart shows the number of sessions produced weekly by the same user, in other words, how many times each user has used our application, on average, a week. For example, during the week of the 30th September there were 533 sessions, which means that the application was opened 533 times during that period.

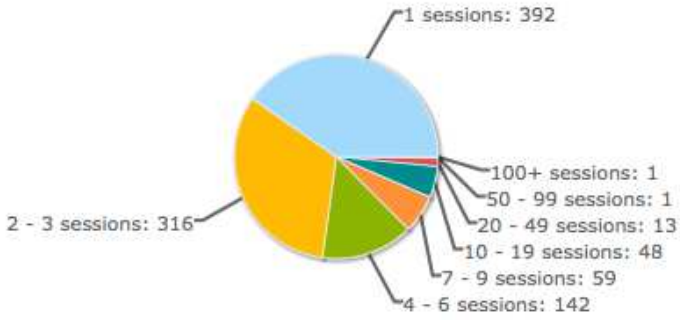


Fig. 14. Session frequency, how many sessions are produced by the same user weekly.

As shown in Figure 14, we have a high percentage of users that quit, or never use, the application once opened for the first time, and as a result we know that we have to work on user retention besides user acquisition.

In order to know how much time users spend on our application, we are tracking session length among other parameters. This is useful to track in which periods of time the application is more used. For example, if an update (new version of the application) has greater session length than the previous one, that new version accomplished its objective. We can observe in Figure 15 that the total time spent by users in the app had a great increase when we updated to version 1.2 (released 11th October) with iOS 7 support. Total time spent on application increased from 3 to 5 hours from previous weeks to 10 hours from the 1.2 version release. The median session length has slightly decreased, that means that there are more sessions per week than in the previous version.

Date	Median Session Length	Total Time Spent in App
11/25/13	40 sec	7 hr 37 min
11/18/13	43 sec	9 hr 5 min
11/11/13	49 sec	10 hr 15 min
11/04/13	44 sec	4 hr 7 min
10/28/13	45 sec	5 hr 17 min
10/21/13	42 sec	3 hr 59 min
10/14/13	46 sec	4 hr 41 min
10/07/13	45 sec	5 hr 44 min
09/30/13	44 sec	8 hr 36 min
09/23/13	41 sec	8 hr 55 min
09/16/13	58 sec	2 hr 35 min

Fig. 15. Median session length and total time spent by users per week.

We can also obtain the time spent per session along with the cumulative time spent, sessions are grouped by predefined periods of time. As shown in Figure 16, users never spend more than 30 minutes in the application, and there are almost 900 sessions which range from 1 to 3 minutes length.

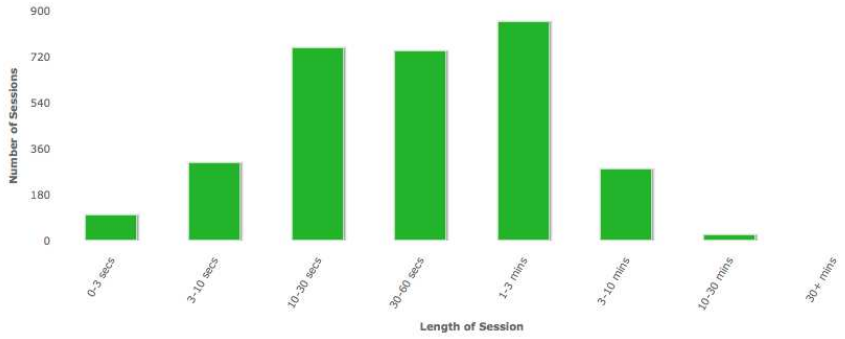


Fig. 16. Number of sessions per length of session along all the time.

In brief, the aim of tracking users is to learn from their behaviour and improve their experience.

5.3 Ranking

In the App Store from Apple, applications are grouped into categories, each application can belong to one or two different categories, these categories include among others: Books, Education, Entertainment, Games, Lifestyle, and Weather, which is the main category for our application. Moreover, applications are grouped according to costumer's country. In our case, the country we are interested in is Spain. With these two parameters we can filter the ranking position for our app, this position depends on the number of downloads. The ranking positions are different for iPhone and iPad, because our application is optimized for being used in iPad, in addition to iPhone.
We can observe in figure 17 the iPhone rank occupied by our application in a given day, the day when our application was in a higher position was 3rd September, and it was the 44th application the most downloaded one that day.

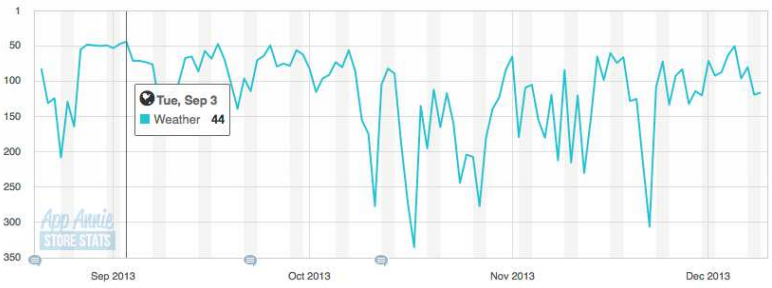


Fig. 17. Ranking for our iPhone application in the Weather category and App Store Spain.

As far as the iPad version is concerned and as it is shown in Figure 18, our application highest position in the ranking was 14th September and it was the 15th most downloaded application that day. It is always above the 250th position and generally above the 150th position, while in iPhone it reached the 300th position.

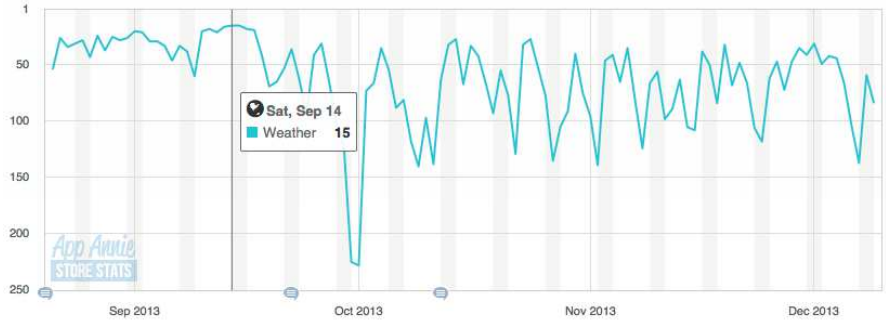


Fig. 18. Ranking for our iPad application in the Weather category and App Store Spain.

6. Conclusions and Future Work

In this work, we successfully combined three fundamental components: data sensors (from direct and indirect users), mobile platforms and spatial proximity operations. As a result, we obtained a mobile application, which allows us to operate in a reliable manner with data from a climate data sensor and from direct users, taking advantage of the geospatial information as well. In fact, this application offers us a valuable type of VGI mashup from different data sources.

We introduced architecture to build this type of applications and we implemented it by means of well-known platforms and libraries. Finally, we analysed the client application by tracking users and sessions, variations in ranking and so forth.

We would like to underline that the climate data sensor we used offers valuable information, but it is provided in a proprietary format included in some comments of the RSS feed. This obstructs its interoperability very much and, in our case, we had to construct a specific service to fix it.

As far as future work is concerned, we are considering including alerts in the application with push notifications. This would be a useful functionality for users who desire to know when a specified station reaches a given temperature, or a given precipitation value.

Moreover, in order to take advantage of the geographical information provided by stations, we would like to perform some spatial analysis such as calculating the optimal location for a new station.

References

[1] M.F. Goodchild. Citizens as sensors: the world of volunteered geography. GeoJournal 69(4): 211-221, 2007.
[2] Gartner Press Release, “Gartner Says Solving ‘Big Data’ Challenge Involves More Than Just Managing Volumes of Data”, June 27, 2011.
[3] D.Z.Sui The wikification of GIS and its consequences: Or Angelina Jolie’s new tattoo and the future of GIS, Computers, Environment and Urban Systems 32 (1), 2008, pp.1–5.
[4] S.Gorman, Why VGI is the Wrong Acronym, http://blog.dc.esri.com/2010/04/15/why-vgi-is-the-wrong-acronym, 2010.
[5] F. Harvey, To Volunteer or to Contribute Locational Information? Towards Truth in Labeling for Crowdsourced Geographic Information, Springer Netherlands, 2013, pp. 31–42.
[6] P.A. Longley [et al.], Geographic Information Systems and Science, Hoboken, NJ, 2011, pp.290–293.
[7] Campbell, A.T., et al., 2008. The Rise of People-Centric Sensing. IEEE Internet Computing 12(4): 12-21.
[8] E. Miluzzo, N. Lane, K. Fodor, R. Peterson, H. Lu, M. Musolesi, S. Eisenman, X. Zheng, and A. Campbell, “Sensing meets mobile social networks: the design, implementation and evaluation of the cenceme application,” in Proceedings of the 6th ACM conference on Embedded network sensor systems. ACM, 2008, pp. 337–350.
[9] Lane, N.D., Miluzzo, E. , Lu, H. , Peebles, D. , Choudhury, T. , Campbell, A.T. A survey of mobile phone sensing. IEEE Communications Magazine Volume 48, Issue 9, September 2010, pp 140-150.
[10] D. Havlik et al., Future Internet enablers for VGI applications. EnviroInfo, page 622-630. Shaker, 2013.
[11] Jiang, Bin, and Lars Harrie. "Cartographic selection using self-organizing maps." ICA Commission on Generalisation and Multiple Representation (2003): 1-7.